

# Documentação de Código

Luiz Henrique de Campos Merschmann  
Departamento de Computação Aplicada  
Universidade Federal de Lavras

[luiz.hcm@ufla.br](mailto:luiz.hcm@ufla.br)



# Na Aula de Hoje



- Documentação de Código
- Javadoc

# Utilizando Classes

Suponha que você está dando manutenção em um sistema implementado por um funcionário aposentado da sua empresa.

- ▶ Em esse sistema, o que não é incomum, consiste em centenas de milhares de linhas de código e milhares de classes.
- ▶ Imagine você ter que ler tudo a fim de entender como a aplicação funciona!
- ▶ Provavelmente seu trabalho será extremamente árduo.



# Utilizando Classes

- ▶ Imagine que você trabalhe em uma grande empresa de desenvolvimento de software.
- ▶ Nesse cenário, é muito comum um grupo de desenvolvedores dividir as tarefas de implementação de um sistema.



Suponha que a parte do sistema que você está implementando precise utilizar a ClasseX que está sendo desenvolvida por outro membro da equipe.

- ▶ Você precisa conhecer a implementação completa da ClasseX para implementar a sua parte?
- ▶ Como dissemos anteriormente, você não deveria ter que conhecer a implementação da ClasseX.
  - ▶ Você precisa conhecer somente a **interface** da ClasseX, ou seja, **as assinaturas dos seus métodos públicos**.

# Utilizando Classes



- ▶ Mas será possível realmente compreender o que os métodos fazem apenas lendo suas assinaturas?
- ▶ O que mais poderia nos ajudar para que não tivéssemos que ler a implementação dos métodos?

## Usar diagrama de classes?

- ▶ Nós ainda aprenderemos a utilizar Diagrama de Classes UML, que ajudam bastante a entendermos a modelagem de um sistema como um todo.
- ▶ No entanto, o diagrama não ajuda a entendermos exatamente o que cada método faz.

# Utilizando Classes



- ▶ Como fazemos quando precisamos utilizar classes das bibliotecas do Java que não conhecemos?
- ▶ Provavelmente você já se deparou com esse problema e utilizou a **documentação das classes** padrão do Java.

É isso mesmo! As **classes** padrão do Java **são documentadas** e você pode acessar essa documentação pela internet.

- ▶ <http://docs.oracle.com/javase/8/docs/api/>



# Documentação API Java

<http://docs.oracle.com/javase/8/docs/api/>

UNIMED INSTITUTO UFLA - Ramais Biblioteca UFLA SIG-UFLA Campus Virtual UFLA Labs-DCC SIGAA - UFLA

Java™ Platform Standard Ed. 8

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

All Classes All Profiles

Packages

java.applet

All Classes

AbstractAction  
AbstractAnnotationValueVisitor6  
AbstractAnnotationValueVisitor7  
AbstractAnnotationValueVisitor8  
AbstractBorder  
AbstractButton  
AbstractCellEditor  
AbstractChronology  
AbstractCollection  
AbstractColorChooserPanel  
AbstractDocument  
AbstractDocument.AttributeContent  
AbstractDocument.Content  
AbstractDocument.ElementEdit  
AbstractElementVisitor6  
AbstractElementVisitor7  
AbstractElementVisitor8

A documentação da API Java pode ser acessada no endereço acima.

**Java™ Platform, Standard Edition 8 API Specification**

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

**Profiles**

- compact1
- compact2
- compact3

**Packages**

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.



# Documentação da Classe String

Screenshot of the Java API documentation for the `String` class. The URL is [docs.oracle.com/javase/8/docs/api/index.html](http://docs.oracle.com/javase/8/docs/api/index.html). The page title is "OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP". The left sidebar shows a list of classes under "java.lang" package, with `String` highlighted and a red arrow pointing to it.

**Class String**

`java.lang.Object`  
`java.lang.String`

All Implemented Interfaces:  
`Serializable, CharSequence, Comparable<String>`

```
public final class String  
extends Object  
implements Serializable, Comparable<String>, CharSequence
```

The `String` class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because `String` objects are immutable they can be shared. For example:

**Essa é a documentação da classe `String`. Nela podemos verificar quais **métodos** estão disponíveis e como eles funcionam.**

**Veja abaixo que logo no início da documentação existe uma descrição geral da classe.**



# Documentação da Classe String

Essa documentação também apresenta um **resumo dos construtores** e de todos os **métodos públicos** disponíveis.



## Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

Deprecated Methods

Modifier and Type	Method and Description
char	<b>charAt(int index)</b> Returns the char value at the specified index.
int	<b>codePointAt(int index)</b> Returns the character (Unicode code point) at the specified index.
int	<b>codePointBefore(int index)</b> Returns the character (Unicode code point) before the specified index.
int	<b>codePointCount(int beginIndex, int endIndex)</b> Returns the number of Unicode code points in the specified text range of this String.
int	<b>compareTo(String anotherString)</b> Compares two strings lexicographically.

# Documentação da Classe String

Temos ainda uma **descrição detalhada** para cada **método**, a qual inclui **exemplos** e detalhamento dos **parâmetros, retorno** e possíveis **exceções**.



## substring

```
public String substring(int beginIndex,  
                      int endIndex)
```

Returns a string that is a substring of this string. The substring begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`. Thus the length of the substring is `endIndex - beginIndex`.

Examples: ←

```
"hamburger".substring(4, 8) returns "urge"  
"smiles".substring(1, 5) returns "mile"
```

Parameters: ←

`beginIndex` - the beginning index, inclusive.

`endIndex` - the ending index, exclusive.

Returns: ←

the specified substring.

Throws: ←

`IndexOutOfBoundsException` - if the `beginIndex` is negative, or `endIndex` is larger than the length of this `String` object, or `beginIndex` is larger than `endIndex`.

# Escrevendo a Documentação de uma Classe

- ▶ Assim como a documentação fornecida para as classes de biblioteca Java, é fundamental documentarmos as classes que implementamos.
- ▶ Desse modo, para alguém utilizar uma determinada classe, basta conhecer a sua interface e a documentação de como ela funciona.

Como documentar nossas próprias classes?

- ▶ O Java fornece uma ferramenta chamada **Javadoc** para nos auxiliar nessa tarefa.

# A ferramenta Javadoc



Quanto trabalho será necessário para produzir essa documentação considerando-se que não tenho conhecimentos de HTML?

Praticamente nenhum trabalho se o seu código estiver devidamente comentado!

- ▶ O Javadoc pode ser utilizado para gerar a documentação a partir do código-fonte.
- ▶ Mais precisamente, a documentação é gerada a partir dos comentários que são colocados no código.
- ▶ O Javadoc transforma os comentários (que devem estar num determinado formato) em documentação HTML.
- ▶ Resumindo, se o código estiver adequadamente comentado, podemos ter facilmente uma documentação como aquela mostrada para a classe String.

# Javadoc – Documentação das Classes

```
import java.util.Random;  
  
/**  
 * Classe que representa uma questão de prova.  
 * Ela fornece o enunciado de uma questão gerada  
 * aleatoriamente, bem como um método para a  
 * correção da mesma.  
 */  
  
public class Questao  
{  
    private static final Random numeroAleatorio = new Random();  
    private String enunciado;  
    private int gabarito;  
    private int idQuestao;  
    private static int totalQuestoes = 0;  
  
    //Elaborando a questão e armazenando a resposta correta  
    public Questao(){
```

Para ser utilizado no Javadoc, basta que o comentário comece com `/**`



# Javadoc – Documentação das Classes

A partir do comentário inserido no código anterior, veja a documentação gerada pela ferramenta Javadoc!



## Class Questao

java.lang.Object  
Questao

public class Questao  
extends java.lang.Object

Classe que representa uma questão de prova. Ela fornece o enunciado de uma questão gerada aleatoriamente, bem como um método para a correção da mesma.



### Constructor Summary

#### Constructors

##### Constructor and Description

Questao()

### Method Summary

#### All Methods

#### Instance Methods

#### Concrete Methods

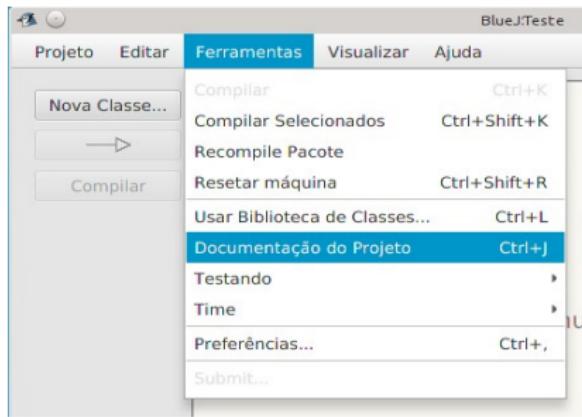
##### Modifier and Type

##### Method and Description

boolean

acertouQuestao(int resposta)

# Javadoc – Como Gerar a Documentação?



Para usar o Javadoc a partir de um terminal, basta digitar a seguinte linha de comando:

**\$ javadoc -d pastaDoc arquivo.java**

Se que todos os seus arquivos .java estiverem na mesma pasta, você pode executar o comando:

**\$ javadoc -d doc \*.java**

Desse modo, a documentação de todos os arquivos será criada e armazenada na pasta *doc*.

As IDEs normalmente têm integração com a ferramenta Javadoc.

No entanto, você também pode usar o Javadoc a partir de um terminal!



# Javadoc – Como Gerar a Documentação?

All Methods	Instance Methods	Concrete Methods
Modifier and Type	Method and Description	
int		getIdQuestao()
int		getNumeroTentativas()
java.lang.String		<b>getStatus()</b>

Usando o comando apresentado anteriormente, os tipos da API Java aparecerão como neste exemplo.

No entanto, você pode enriquecer sua documentação acrescentando links para documentação oficial do Java. Para isso, basta acrescentar na linha de comando a opção `-link`, como mostrado a seguir:

```
$ javadoc -link https://docs.oracle.com/javase/8/docs/api/ -d doc *.java
```

All Methods	Instance Methods	Concrete Methods
Modifier and Type	Method and Description	
int		getIdQuestao()
int		getNumeroTentativas()
String		<b>getStatus()</b>

Usando a opção `-link`, podemos clicar sobre os tipos da API Java para acessarmos a sua documentação oficial.

# Javadoc – Documentação dos Métodos

Também podemos colocar comentários nos **métodos públicos** para termos uma documentação completa!



## Method Detail

### getEnunciado

```
public String getEnunciado()
```

Método que retorna o enunciado de uma questão gerada aleatoriamente.

### getIdQuestao

```
public int getIdQuestao()
```

Método que retorna o identificador da questão.

### acertouQuestao

```
public boolean acertouQuestao(int resposta)
```

Método para corrigir uma questão. Ele recebe uma resposta do usuário e a compara com o gabarito da questão, retornando true se a resposta estiver correta ou false caso contrário.

```
/**  
 * Método para corrigir uma questão.  
 * Ele recebe uma resposta do usuário e a compara  
 * com o gabarito da questão, retornando true se  
 * a resposta estiver correta ou false caso contrário.  
 */  
public boolean acertouQuestao(int resposta){  
    if (resposta == gabarito)  
        return true;  
    else  
        return false;  
}
```



# Javadoc – Palavras-chave

Além dos comentários gerais, podemos usar também as **palavras-chave**.



```
/**  
 * Método para corrigir uma questão.  
 * Ele recebe uma resposta do usuário e a compara  
 * com o gabarito da questão.  
 *  
 * @param resposta inteiro contendo a resposta do usuário  
 * @return boolean - true se a resposta estiver  
 * correta ou false caso contrário.  
 */  
public boolean acertouQuestao(int resposta){  
    if (resposta == gabarito)  
        return true;  
    else  
        return false;  
}
```

## acertouQuestao

public boolean acertouQuestao(int resposta)

Método para corrigir uma questão. Ele recebe uma resposta do usuário e a compara com o gabarito da questão.

### Parameters:

resposta - inteiro contendo a resposta do usuário

### Returns:

boolean - true se a resposta estiver correta ou false caso contrário.

As palavras-chave sempre começam com `@` e indicam elementos específicos no comentário.

Observe a documentação gerada a partir das palavras-chave `@param` e `@return`.

# Javadoc – Palavras-chave

- ▶ **@author**: Nome do autor de uma classe.
- ▶ **@version**: Número de versão de uma classe.
- ▶ **@param**: Parâmetro de um método.
- ▶ **@return**: Retorno de um método.
- ▶ **@throws**: Indica exceção lançada pelo método.

# Javadoc – Verificação de Erros

- A ferramenta Javadoc também verifica se existem erros na documentação (assim como o compilador faz com o código).

Exemplo 1: Classe sem visibilidade gera erro.

```
Loading source file Questao.java...
Constructing Javadoc information...
javadoc: error - No public or protected classes found to document.
1 error
```

Exemplo 2: Falta de documentação de parâmetros e/ou retorno gera aviso.

```
Generating doc/Questao.html...
Questao.java:39: warning: no @return
    public String getEnunciado(){
                           ^
Questao.java:46: warning: no @return
    public int getIdQuestao(){
                           ^
```

# Javadoc – Uso Durante a Implementação

Veja mais uma utilidade da documentação! As informações sobre métodos e classes que **visualizamos nas IDEs** quando estamos programando são provenientes dessa documentação.



The screenshot shows a Java code editor with the following code:

```
58     Scanner entrada = new Scanner(System.in);
59
60     System.out.println("Digite o autor:");
61     autor = entrada.nextLine();
62
63     System.out.println("Digite a mensagem de texto:");
64     mensagem = entrada.nextLine();
65
66     feed.posta
67
68     entrada
69 }
70
71 private s
72 {
73     String
```

A tooltip is displayed over the `feed.posta` call, showing the Javadoc for the `posta` method:

Método para postar uma mensagem com foto no Feed de Notícias. Ela recebe o autor, o caminho do arquivo que contém a foto e a legenda da foto. Obs: como o Feed de Notícias é exibido em linha de comando, o caminho da foto é exibido ao invés da foto.

**Parameters:**

- `autor` String com o nome do autor da mensagem
- `arquivoFoto` Caminho (pasta/arquivo) onde se encontra a foto no computador
- `legenda` Legenda a ser exibida junto com a foto

Press 'Ctrl+Space' to show Template Proposals  
Press 'Tab' from proposal table or click for focus

Fonte: Material do Prof. Júlio.

# Documente o seu Código!

- ▶ **Sempre documente** o seu código no formato do **Javadoc**.
- ▶ Faça comentários **relevantes** e insira **exemplos de uso**.
- ▶ Lembre-se que **quem vai usar** sua classe **não conhece a implementação** da mesma.
- ▶ Os comentários devem ter um **nível de detalhamento** que permita a qualquer um **utilizar** os seus métodos **sem precisar ler a implementação** dos mesmos.
- ▶ Utilize como **referência** a documentação das **classes da API Java**.

# Perguntas?

